Learning URI Selection Criteria to Improve the Crawling of Linked Open Data (Extended Abstract)

Hai Huang, Fabien Gandon

Learning URI Selection Criteria to Improve the Crawling of Linked Open Data (Extended Abstract)*

Hai Huang and **Fabien Gandon**
Inria, Université Côte d'Azur, CNRS, I3S, France
{hai.huang, fabien.gandon}@inria.fr

Abstract

A Linked Data crawler performs a selection to focus on collecting linked RDF (including RDFa) data on the Web. From the perspectives of throughput and coverage, given a newly discovered and targeted URI, the key issue of Linked Data crawlers is to decide whether this URI is likely to dereference into an RDF data source and therefore it is worth downloading the representation it points to. Current solutions adopt heuristic rules to filter irrelevant URIs. But when the heuristics are too restrictive this hampers the coverage of crawling. In this paper, we propose and compare approaches to learn strategies for crawling Linked Data on the Web by predicting whether a newly discovered URI will lead to an RDF data source or not. We detail the features used in predicting the relevance and the methods we evaluated including a promising adaptation of FTRL-proximal online learning algorithm. We compare several options through extensive experiments including existing crawlers as baseline methods to evaluate their efficiency.

1 Introduction

Linked Data extends the principles of the World Wide Web from linking documents to that of linking pieces of data to weave a Web of Data. This relies on the well-known linked data principles [Heath and Bizer, 2011; Berners-Lee, 2006] including the use of HTTP URIs that can be dereferenced and the provision of useful and linked descriptions upon access so that we can discover more things.

A large amount of data are now being made available as linked data in various domains such as health, publication, agriculture, music, etc., and the Web of Linked Data is growing exponentially [Ermilov *et al.*, 2016]. In order to harvest this enormous data repository, crawling techniques for Linked Data are becoming increasingly important. Different from conventional crawlers, crawling for Linked Data is performed selectively to collect structured data connected by RDF links. The design objective of Linked Data crawlers is to fetch linked RDF data in different formats – RDF/XML, N3, RDFa, JSON-LD, etc. – as much as possible within a reasonable time while minimizing the download of irrelevant URIs – i.e. leading to resources without RDF content. Therefore our challenge is to identify as soon as possible the URIs referencing RDF sources without downloading them first. Our research question here is: *can we learn efficient URI selection criteria to identify sources of Linked Open Data*?

To solve this problem, we propose and compare methods on real data to predict whether a newly discovered URI will lead to RDF data source or not. We extract information from the targeting and referring URI and from the context (RDF data graph) where URIs and their links were discovered in order to produce features fed to learning algorithms and in particular to an FTRL-proximal online method employed to build the prediction model.

The contributions of this work include: 1) we identify the features to predict whether a target URI will lead to some RDF data or not; 2) we adapt the FTRL-proximal algorithm to our task and build an online prediction model; and 3) we implement a Linked Data crawler with the online prediction model and evaluate its performance.

2 Related Work

Semantic web crawlers differ from traditional web crawlers in only two aspects: the format of the source (RDF format) it is traversing, and the means to link RDF across data sources. There exist some work [Dodds, 2006; Isele *et al.*, 2010; Hogan *et al.*, 2011] in the field of Semantic Web/Linked Data crawling. The two main representative crawlers for Linked Data are LDSpider [Isele *et al.*, 2010] and SWSE crawler [Hogan *et al.*, 2011]. They crawl the Web of Linked Data by traversing RDF links between data sources and follow the Linked Data principles [Heath and Bizer, 2011; Berners-Lee, 2006].

In order to reduce the amount of HTTP lookups and downloading wasted on URIs referencing non-RDF resources, these previous works apply heuristic rules to identify relevant URIs [Isele *et al.*, 2010; Hogan *et al.*, 2011]. The URIs with common file extensions (e.g., html/htm, jpg, pdf, etc.) and those without appropriate HTTP Content-Type Header

*This is an abridged version of the paper [Huang and Gandon, 2019] that won the best paper award at ESWC-2019. This work is supported by the ANSWER project PIA FSN2 N°P159564-2661789/DOS0060094 between Inria and Qwant.

(such as application/rdf+xml) are classified as non-RDF content URIs. The content of these URIs would not be retrieved by these crawlers. Although this heuristic-based method is efficient, it impairs the recall of the crawling. This method makes the assumption that data publishers have provided correct HTTP Content-Type Header but this is not always the case. It can happen that the server does not provide the desired content type. Moreover, it may happen that the server returns an incorrect content type. For example, Freebase did not provide any content negotiation or HTML resource representation [Färber *et al.*, 2018], and only text/plain was returned as content type. In [Hogan *et al.*, 2010], it is reported that 17% of RDF/XML documents are returned with a content-type other than application/rdf+xml. As a result, a huge volume of RDF data is missed by these methods.

Traditional focused crawlers on the Web aim to crawl a subset of Web pages based on their relevance to a specific topic [Chakrabarti *et al.*, 1999; Diligenti *et al.*, 2000]. These methods build a relevancy model typically encoded as a classifier to evaluate the web documents for topical relevancy. Our work here is different in the sense that we do not filter on the topics but on the type of content. Meusel et al. [Meusel *et al.*, 2014] proposed a focused crawling approach to fetch microdata embedded in HTML pages. Umbrich et al. [Umbrich *et al.*, 2008] built a crawler focused on gathering files of a particular media type and based on heuristics.

3 Prediction Model for Crawling Criteria

In this section, we present the prediction task, the feature sets extracted for the task of prediction and then describe the prediction model based on FTRL-proximal online learning algorithm.

3.1 Task Description

Since the task of Linked Data crawlers is to fetch RDF data on the Web, we are interested in RDF-relevant URIs:

Definition 1. ***(RDF-Relevant)*** *Given a URI u, we consider that u is RDF-relevant if the representation obtained by dereferencing u contains RDF data. Otherwise, u is called non RDF-relevant. We note U^R the set of RDF relevant URIs and U^I the set of non RDF-relevant URIs with $U = U^R \dot{\cup} U^I$.*

For the URIs that have certain file extensions such as *.rdf/owl or HTTP Content Types Headers such as application/rdf+xml, text/turtle, etc., it is trivial to know the RDF-relevance of them. In this work, we focus on a knid of URIs called hard URIs whose RDF-relevance *cannot* be known by these heuristics.

Definition 2. ***(Hard URI)*** *We call u a hard URI if the RDF relevance of u cannot be known straightforwardly by its file extension or HTTP Content-Type Header.*

For example, URI u with HTTP Content-Type header text/html is a hard URI since RDFa data could be embedded in u. As reported in [Färber *et al.*, 2018], the URIs with HTTP Content-Type Header text/plain may contain RDF data so they are hard URIs too.

Then, for our prediction task, we consider four types of URIs involved in the prediction: the target URI, the referring URI, direct property URIs and sibling URIs.

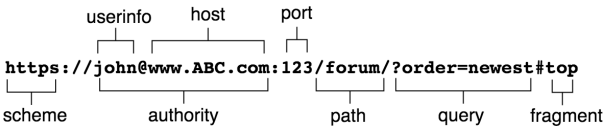

Figure 1: Example of a URI with component parts.

Definition 3. ***(Target URI)*** *We call target URI and note u_t the URI for which we want to predict if the $deref(u_t)$ will lead to a representation that contains RDF data i.e. if u_t is RDF-Relevant.*

Definition 4. ***(Context RDF Graph)*** *Given an RDF relevant URI u_r containing the RDF graph G_t^r, we define $G_t^r = (V, E)$ as the context RDF graph of URI u_t if u_t appears in G_t^r as a node, i.e., $u_t \in V$.*

Definition 5. ***(Referring URI)*** *Given a target URI u_t, we call "referring URI" and note u_r the RDF-relevant URI that was dereferenced into a representation $deref(u_r)$ containing the context RDF graph G_t^r in which we found the target URI u_t.*

Definition 6. ***(Sibling Set)*** *The sibling set of u_t in the context RDF Graph G_t^r denoted by Sib_{u_t} is the set of all other URIs that have common subject-property or property-object pair with u_t in G_t^r. Sib_{u_t} is formally defined as:*

$$Sib_{u_t} = \{s | \exists p, o \in G_t^r, s.t. (u_t, p, o) \in G_t^r \wedge (s, p, o) \in G_t^r\} \vee \{o | \exists s, p \in G_t^r, s.t. (s, p, u_t) \in G_t^r \wedge (s, p, o) \in G_t^r\}$$

Definition 7. ***(Direct Property Set)*** *The direct property set PS_t is the set of properties that connect u_t in the context graph G_t^r:*

$$PS_t = \{p | \exists s, o \in G_t^c, s.t. (u_t, p, o) \in G_t^r \vee (s, p, u_t) \in G_t^r\}$$

Prediction Task. Using these definitions we can now define our prediction task. Suppose that a hard URI u_t is discovered from the context graph G_t^r obtained by dereferencing a referring URI u_r. We want to predict if u_t is RDF-relevant based on some features extracted from u_t, u_r, PS_t and Sib_{u_t}. Our task is to learn the mapping $Relevant : U \rightarrow \{0, 1\}$ with $Relevant(u)$ equals 1 if u is RDF-relevant ($Relevant|_{U^R} \mapsto 1$) and equals 0 if u is not RDF-relevant ($Relevant|_{U^I} \mapsto 0$)

3.2 Feature Extraction

We distinguish between two kinds of features that can be exploited for the prediction intrinsic and extrinsic.

Definition 8. ***(Intrinsic URI Features)*** *The intrinsic features of a URI u are features obtained by an extraction $F_{int} : U \rightarrow F$ that relies exclusively on the URI identifier itself.*

An example of an intrinsic feature is the protocol e.g. `http`.

Definition 9. ***(Extrinsic URI Features)*** *The extrinsic features of a URI u are features obtained by performing some network call on the URI $F_{ext} : U \rightarrow F$.*

The URI generic syntax consists of a hierarchical sequence of several components. An example of URI is shown in Fig.1.

The intrinsic features $F_{int}(u)$ consider that the different components of a URI u are informative and contain helpful

information for prediction. The components include: scheme, authority, host, path, query, fragment information. We generate the intrinsic features of u based on these components.

We further distinguish two kinds of extrinsic features:

Definition 10. *(**URI Header Features**) These extrinsic features of a URI u are obtained by an `HTTP HEAD` call $F_{head} : U \rightarrow F$ and do not require to download the representation of the resource identified by u.*

Definition 11. *(**URI Representation Features**) These extrinsic features of a URI u are obtained by an `HTTP GET` call $F_{get} : U \rightarrow F$ and characterize the content obtained after a complete download of the representation of the resource identified by u.*

We distinguish F_{head} and F_{get} because they have different costs in terms of network access time. The URI header feature we will consider in this paper is the content type e.g. feature_u_t_contentType='text/html'.

URI representation features come from the content of the referring URI u_r, namely the context graph G_t^r of u_t which includes the direct properties and siblings information of u_t.

Definition 12. *(**URI Similarity Features**) The similarity between two URIs u_s and u_t denoted by $simValue(u_s, u_t)$ is defined using the Levenshtein distance[Levenshtein, 1966] between the two strings representing these URIs. In order to reduce the feature space, a threshold τ is set for the similarity value and if the similarity value is larger than τ, we discretize the similarity as $simValue(u_t, u_s) = high$ and otherwise $simValue(u_t, u_s) = low$.*

Definition 13. *(**RDF Relevance Features**) The boolean characteristic of being RDF-relevant for a URI u is noted $F_{RDFrel}(u)$ and returns true if u is RDF-revelant and false otherwise.*

With the types of features explained above we can now define four atomic feature sets based on the sources the features come from to explore and evaluate when training and predicting the RDF-relevance of a target URI u_t:

- $F_{+t} = F_{int}(u_t) + F_{head}(u_t)$ is a feature set considering the intrinsic and header features of the target URI u_t.
- $F_{+r} = F_{int}(u_r)$ is a feature set considering the intrinsic features of the referring URI u_r.
- $F_{+p} = \bigcup_{p \in PS_t} F_{int}(p)$ is the set of intrinsic features of each direct property of the target URI u_t.
- $F_{+x} = \bigcup_{u_s \in Sib_{u_t}} F_x(u_s)$ is a feature set including feature crosses that combine the intrinsic, header, similarity and relevance features of the sibling URIs of u_t. This supports predictive abilities beyond what those features can provide individually and can be interpreted as using a logical conjunction 'AND' to combine these features $F_x(u_s) = F_{int}(u_s) \times F_{head}(u_s) \times F_{sim}(u_s, u_t) \times F_{RDFrel}(u_s)$

With these definitions we can now consider and evaluate any combination of feature sets. We note F_{+a} the feature set with a being a combination of the atomic feature sets as defined above. For instance an experiment using the feature set F_{+t+r} will only consider as inputs the intrinsic and header features of the target URI u_t and the intrinsic feature of referring URI u_r. In Section 4, the predictive abilities of different combinations of the feature sets are examined.

Algorithm 1: Online prediction with the FTRL-Proximal algorithm

Data: URIs $u_1, u_2, \cdots, u_T$
Result: $\hat{y}_1, \cdots, \hat{y}_T$
for $t = 1$ *to T do* **do**
  get feature vector $\mathbf{x}_t$ of u_t;
  probability $p_t = sigmoid(\mathbf{x}_t \cdot \mathbf{w}_t)$;
  if $p_t > \tau$ **then**
    output $\hat{y}_t$=1;
  else
    output $\hat{y}_t$=0;
  end
  observe real label $y_t \in \{0, 1\}$;
  update $\mathbf{w}_{t+1}$ by equation (1);
end

3.3 Online Prediction

During the process of crawling, the crawler will encounter URIs belonging to millions of domains, and the number of properties could be tens of thousands. Obviously, the potential feature space will be huge. Thus, we use feature hashing technique [Weinberger *et al.*, 2009] to map high-dimensional features into binary vectors.

We assume that data becomes available in a sequential order during the crawling process. The predictor makes a prediction at each step and can also update itself: it operates in an online style. Compared to batch methods, online methods are more appropriate for the task of crawling since the predictor has to work in a dynamic learning environment. FTRL-Proximal [McMahan *et al.*, 2013; McMahan, 2011] developed by Google has been proven to work well on the massive online learning problems. FTRL-proximal outperforms the other online learning algorithms in terms of accuracy and sparsity, and has been widely used in industry, e.g., recommender systems and advertisement systems. We adopt this learning algorithm in our work.

We use $\mathbf{x}_t$ to denote the feature vector of URI u_t and $y_t \in \{0, 1\}$ the true class label of u_t. Given a sequence of URIs $u_1, u_2, \cdots, u_r, \cdots, u_t$, the process of online prediction based on FTRL-Proximal algorithm is shown in Algorithm 1. We adapted the original algorithm to make it output binary values by setting a decision threshold τ. If the predicted probability p_t is greater than the decision threshold $\tau \in [0, 1]$, it outputs prediction $\hat{y}_t = 1$; otherwise $\hat{y}_t = 0$.

At round $t + 1$, the FTRL-Proximal algorithm uses the update formula (1):

$$\mathbf{w}_{t+1} = argmin_{\mathbf{w}}(\sum_{s=1}^{t} \mathbf{w} \cdot \mathbf{g}_s + \frac{1}{2} \sum_{s=1}^{t} \sigma_s \parallel \mathbf{w} - \mathbf{w}_s \parallel_2^2 + \lambda_1 \parallel \mathbf{w} \parallel_1). \quad (1)$$

In equation (1), $\mathbf{w}_{t+1}$ is the target model parameters to be updated in each round. In the first item of equation (1), $\mathbf{g}_s$

Combination of Feature Sets	KNN		Naive Bayes		SVM	
	F-measure	Accuracy	F-measure	Accuracy	F-measure	Accuracy
$F_{+t+r+p+x}$	0.6951	0.7407	0.7154	0.7462	0.7944	0.7722
F_{+t+p+x}	0.6261	0.6832	0.7094	0.7413	0.7801	0.7643
F_{+t+r+x}	0.6773	0.7121	0.7111	0.7448	0.7829	0.7650
F_{+t+r+p}	0.7592	0.7731	0.7660	0.7701	**0.8216**	**0.7902**
F_{+r+p+x}	0.6015	0.7010	0.6328	0.7075	0.6839	0.7074
F_{+t+x}	0.5582	0.6912	0.6012	0.6172	0.6828	0.6277
F_{r+p}	0.3953	0.5810	0.4874	0.6097	0.6790	0.6424
F_{+p+x}	0.4392	0.5739	0.6086	0.6238	0.6689	0.6269

(a)

Crawler	Percentage
BFS	0.302
crawler_NB (20K)	0.341
crawler_NB (40K)	0.345
crawler_SVM (20K)	0.402
crawler_SVM (40K)	0.413
crawler_KNN (20K)	0.331
crawler_KNN (40K)	0.324
LDCOC ($\tau = 0.5, \epsilon = 0.17$)	**0.655**

(b)

Figure 2: (a) Performance of the combinations of feature sets; (b) Percentage of retrieved RDF relevant URIs by different crawlers.

is the gradient of loss function for training instance s. The second item of equation (1) is a smoothing term which aims to speed up convergence and improve accuracy, and σ_s is a non-increasing learning rate defined as $\sum_{s=1}^{t} \sigma_s = \sqrt{t}$. The third item of equation (1) is a convex regularization term used to prevent over-fitting and induce sparsity.

Subsampling. Not all URIs are considered as training instances since we are interested in hard URIs. We exclude from training URIs with the extensions such as *.rdf/owl. Inversely, URIs with the file extension *.html/htm are included in the training set since they may contain RDFa data. The true class label y_t is required to update the predictor online. In our scenario, to observe the true class label of a URI we have to download it and check whether it contains RDF data or not. We cannot afford to download all URIs because of the network overhead, and our target is to build a prediction model that avoids downloading unnecessary URIs. There, an appropriate subsampling strategy is needed. We found that the positive URIs are rare and relatively more valuable. For each round, if URI u_t is predicted positive namely $\hat{y}_t = 1$, we retrieve the content of u_t and observe the real class label y_t. Then the predictor can be updated by new training instance $(y_t, \mathbf{x_t})$. For those URIs predicted negative, we only select a fraction $\epsilon \in]0, 1]$ of them to download and observe their true class label. Here ϵ is a balance between online prediction precision (which requires as many URIs as possible for online training) and downloading overhead. To deal with the bias of this subsampled data, we assign an importance weight $\frac{1}{\epsilon}$ to these examples.

4 Empirical Evaluation

In the first experiment, we evaluate the predictive ability of different combinations of feature sets introduced in Section 3.2 by using offline classifiers including SVM, KNN and Naive Bayes. The dataset including 103K URIs and 9,825 different hosts is generated by operating a Breadth-First Search (BFS) crawl. As described in Section 3.2, the feature sets include F_{+t} derived from the target URI u_t, F_{+r} derived from the referring URI u_r, F_{+p} derived from the direct properties of u_t and F_{+x} derived from the siblings of u_t.

In Figure 2(a) we report the results for the 4-element combination of feature sets ($F_{+t+r+p+x}$), all 3-element combinations and the 2-element combinations (F_{+t+x}, F_{r+p}, F_{+p+x}) which have the best performance in their class. Among all combinations, the 3-element combination F_{+t+r+p} outperforms the other combinations with F-measure 0.8216 and accuracy 0.7902. The 4-element combination $F_{+t+r+p+x}$ as the second best combination scores F-measure 0.7944 and accuracy 0.7722. We found that augmenting sibling features to F_{+t+r+p} is not helpful to improve the performance in the cases of three classifiers. We also found that the performance of F_{+r+p+x} which is derived by excluding feature set F_{+t} from $F_{+t+r+p+x}$ decreases a lot (the worst in all 3-element combinations) compared to the performance of $F_{+t+r+p+x}$. It indicates that the features from target URI u_t are important.

In the next experiment, we evaluate the performance of the proposed online prediction method against several baseline methods. We implemented the proposed crawler denoted by **LDCOC** (Linked Data Crawler with Online Classifier). The implementation detail of LDCOC is provided in [Huang and Gandon, 2019]. The decision threshold τ in Algorithm 1 is set to 0.5 and the parameter ϵ of LDCOC is set to 0.17 according to the ratio of the number of positive URIs to the number of negative URIs in the previous training set. We also implemented three crawlers with offline classifiers including SVM, KNN and Naive Bayes to select URIs. The classifiers are pre-trained with two training sets (with size 20K and 40K). The BFS crawler is another baseline to be compared to. As suggested above, we use the feature set F_{+t+r+p} for the experiment. Figure 2(b) shows the percentage of retrieved RDF relevant URIs by different crawlers after crawling 300K URIs. Since the Linked Data Web is a dynamic environment, the crawlers with offline classifiers pre-trained by a small size training set would not improve performance a lot and our proposed crawler LDCOC outperforms these crawlers.

5 Conclusion

We have presented a solution to learn URI selection criteria to improve the crawling of Linked Open Data. The method is able to predict whether a newly discovered URI contains RDF content or not by extracting features from several sources and building a prediction model based on FTRL-proximal learning algorithm. The experimental results demonstrate that the coverage of the crawl is improved compared to baseline methods. Our method can also be generalized to crawl other kinds of Linked Data such as JSON-LD, Microdata, etc.

References

[Berners-Lee, 2006] Tim Berners-Lee. Linked data - design issues. 2006.

[Chakrabarti *et al.*, 1999] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[Diligenti *et al.*, 2000] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *VLDB*, pages 527–534, 2000.

[Dodds, 2006] Leigh Dodds. Slug : A Semantic Web Crawler. 2006.

[Ermilov *et al.*, 2016] Ivan Ermilov, Jens Lehmann, Michael Martin, and Sören Auer. Lodstats: The data web census dataset. In *International Semantic Web Conference (2)*, volume 9982 of *Lecture Notes in Computer Science*, pages 38–46, 2016.

[Färber *et al.*, 2018] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and YAGO. *Semantic Web*, 9(1):77–129, 2018.

[Heath and Bizer, 2011] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web Into a Global Data Space*, volume 1. Morgan & Claypool Publishers, 2011.

[Hogan *et al.*, 2010] Aidan Hogan, Andreas Harth, Alexandre Passant, Stefan Decker, and Axel Polleres. Weaving the pedantic web. In *LDOW*, 2010.

[Hogan *et al.*, 2011] Aidan Hogan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. Searching and browsing linked data with SWSE: the semantic web search engine. *J. Web Sem.*, 9(4):365–401, 2011.

[Huang and Gandon, 2019] Hai Huang and Fabien Gandon. Learning URI selection criteria to improve the crawling of linked open data. In *ESWC*, pages 194–208, 2019.

[Isele *et al.*, 2010] Robert Isele, Jürgen Umbrich, Christian Bizer, and Andreas Harth. Ldspider: An open-source crawling framework for the web of linked data. In *Proceedings of the ISWC 2010 Posters & Demonstrations Track*, 2010.

[Levenshtein, 1966] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. Dokl.*, 6:707–710, 1966.

[McMahan *et al.*, 2013] H. Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: a view from the trenches. In *SIGKDD*, pages 1222–1230, 2013.

[McMahan, 2011] H. Brendan McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and L1 regularization. In *AISTATS*, pages 525–533, 2011.

[Meusel *et al.*, 2014] Robert Meusel, Peter Mika, and Roi Blanco. Focused crawling for structured data. In *CIKM*, pages 1039–1048, 2014.

[Umbrich *et al.*, 2008] Jürgen Umbrich, Andreas Harth, Aidan Hogan, and Stefan Decker. Four heuristics to guide structured content crawling. In *ICWE*, pages 196–202, 2008.

[Weinberger *et al.*, 2009] Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *ICML*, pages 1113–1120, 2009.